Name: ISHIMWE Elie **10/10/23**
ID: 24783

JAVA ASSIGNMENT

Q1. Difference between parameter and arguments.

Ans/

Parameters and arguments are terms often used in the context of functions or methods in programming:

Parameters:

Parameters are placeholders or variables defined in the function's declaration or definition.
They are used to specify what kind of data or information a function expects when it is called.
Parameters act as variables that are local to the function, and their values are determined when the function is called.
Parameters help define the function's interface, indicating what input it requires.

Arguments:

Arguments are the actual values or data that you pass into a function when you call it.
They are used to supply the values for the parameters of a function.
Arguments provide the concrete data that the function will work with when it is executed.
The number and order of arguments must match the number and order of parameters in the function's definition.

Q2. Differentiate SQL and Non-SQL with examples.

ANS/

SQL (Structured Query Language) and NoSQL (Not Only SQL) are two different approaches to managing and manipulating data in databases. Here's a differentiation between SQL and NoSQL databases along with examples for each:

SQL (Relational Databases):

SQL databases are also known as Relational Database Management Systems (RDBMS).
They use a structured schema with tables, rows, and columns to organize and store data.

SQL databases are suitable for structured and well-defined data.
They use SQL as the query language for data manipulation.

Example SQL database: MySQL
Example SQL query:
SQL
SELECT * FROM customers WHERE city = 'New York';

NoSQL (Non-Relational Databases):

NoSQL databases are more flexible and can handle unstructured or semi-structured data.
They don't require a fixed schema and can store data in various formats like JSON, XML, or key-value pairs.
NoSQL databases are suitable for large-scale, distributed, and dynamic data.
Example NoSQL databases:
Document-based: MongoDB
Key-Value: Redis
Column-family: Apache Cassandra
Graph-based: Neo4j
Example NoSQL query (MongoDB):
javascript
// Insert a document into a MongoDB collection
db.users.insertOne({
name: "John",
age: 30,
city: "Los Angeles"
});

Q3. Why does execute method return false although the process is successful? And when does it return true?

ANS/

In the context of programming and APIs, the behavior of the execute method can vary depending on the specific library, framework, or class you are using. However, I can provide some general insights:

Execute Method Returning False:

The execute method often returns false when an operation or process fails or encounters an error.

It is a common practice in many programming libraries to return false to indicate that something went wrong during the execution.

The exact reasons for a false return value can vary but may include issues like invalid input, resource constraints, or network errors.

Execute Method Returning True:

Conversely, the execute method may return true to indicate that the operation or process was successful and completed without errors.

In this case, a true return value typically signifies that the requested task has been carried out as intended.

To determine the specific conditions under which the execute method returns true or false, you should refer to the documentation or source code of the library or class you are using. It's essential to handle the return values appropriately in your code to account for both successful and failed operations. Additionally, some libraries or methods might use different conventions, such as throwing exceptions instead of returning Boolean values, to indicate errors, so it's crucial to be aware of the error-handling mechanisms in your programming environment.

Q4. How to validate (book_id).

To validate a book ID in Java, you need to define the rules or criteria that a valid book ID must meet, and then write code to check if a given book ID adheres to those rules. Here's a general outline of how you can do it:

Define the criteria for a valid book ID:

Length: Determine the required length of the book ID.

Format: Specify any specific format or pattern the ID must follow.

Characters: Define the allowable characters (e.g., alphanumeric, dashes, underscores).

Write a validation function:

Create a Java method that takes a book ID as input and returns a boolean value indicating whether it's valid.

Implement the validation logic:

Within the validation function, check if the book ID meets the defined criteria. You can use regular expressions, string manipulation functions, or a combination of methods to perform these checks.

Here's a simplified example in Java to demonstrate the concept:

```
public class BookIDValidator {


   public static boolean isValidBookID(String bookID) {
      // Define criteria (e.g., length, format, characters)
      int requiredLength = 10;
      String allowedCharacters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
```

```
        // Check length
        if (bookID.length() != requiredLength) {
            return false;
        }

        // Check format (e.g., uppercase letters and digits only)
        for (char character : bookID.toCharArray()) {
            if (!allowedCharacters.contains(String.valueOf(character))) {
                return false;
            }
        }

        // Additional checks can be added as needed

        // If all criteria pass, the book ID is valid
        return true;
    }
      public static void main(String[] args) {
        // Test the isValidBookID function
        String bookID1 = "AB12345678";
        String bookID2 = "InvalidID";
        System.out.println("Book ID 1 is valid: " + isValidBookID(bookID1));
        System.out.println("Book ID 2 is valid: " + isValidBookID(bookID2));
    }
}
```